Grouping_dataset

November 27, 2024

1 Group by Mechanics

During data analysis, it is often essential to cluster or group data together based on certain criteria. For example, an e-commerce store might want to group all the sales that were done during the Christmas period or the orders that were received on Black Friday. These grouping concepts occur in several parts of data analysis.

While working with the pandas dataframes, our analysis may require us to split our data by certain criteria. Groupby mechanics amass our dataset into various classes in which we can perform exercises and make changes, such as the following: 1. Grouping by features, hierarchically 2. Aggregating a dataset by groups 3. Applying custom aggregation functions to groups 4. Transforming a dataset groupwise

The pandas groupby method performs two essential functions: 1. It splits the data into groups based on some criteria. 2. It applies a function to each group independently.

To work with groupby functionalities, we need a dataset that has multiple numerical as well as categorical records in it so that we can group by different categories and ranges.

[10]:
```
import pandas as pd
```

[13]:
```
df = pd.read_csv("automobileEDA.csv")
df.head(10)
```

[13]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	\
0	3	?	alfa-romero	gas	std	two	
1	3	?	alfa-romero	gas	std	two	
2	1	?	alfa-romero	gas	std	two	
3	2	164	audi	gas	std	four	
4	2	164	audi	gas	std	four	
5	2	?	audi	gas	std	two	
6	1	158	audi	gas	std	four	
7	1	?	audi	gas	std	four	
8	1	158	audi	gas	turbo	four	
9	0	?	audi	gas	turbo	two	

	body-style	drive-wheels	engine-location	wheel-base	…	engine-size	\
0	convertible	rwd	front	88.6	…	130	
1	convertible	rwd	front	88.6	…	130	

```
2    hatchback          rwd           front        94.5  …          152
3        sedan          fwd           front        99.8  …          109
4        sedan          4wd           front        99.4  …          136
5        sedan          fwd           front        99.8  …          136
6        sedan          fwd           front       105.8  …          136
7        wagon          fwd           front       105.8  …          136
8        sedan          fwd           front       105.8  …          131
9    hatchback          4wd           front        99.5  …          131

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000       21
1         mpfi  3.47    2.68               9.0        111      5000       21
2         mpfi  2.68    3.47               9.0        154      5000       19
3         mpfi  3.19     3.4              10.0        102      5500       24
4         mpfi  3.19     3.4               8.0        115      5500       18
5         mpfi  3.19     3.4               8.5        110      5500       19
6         mpfi  3.19     3.4               8.5        110      5500       19
7         mpfi  3.19     3.4               8.5        110      5500       19
8         mpfi  3.13     3.4               8.3        140      5500       17
9         mpfi  3.13     3.4               7.0        160      5500       16

  highway-mpg  price
0          27  13495
1          27  16500
2          26  16500
3          30  13950
4          22  17450
5          25  15250
6          25  17710
7          25  18920
8          20  23875
9          22      ?

[10 rows x 26 columns]
```

As you can see there are multiple columns with categoical variable. Using groupby() function lets group these data set on the basis of body-style.

[16]:
```
#groups dataframe according to body-style and prints the keys of groups
df.groupby('body-style').groups.keys()
```

[16]:
```
dict_keys(['convertible', 'hardtop', 'hatchback', 'sedan', 'wagon'])
```

[22]:
```
df.groupby('body-style').get_group("convertible")
```

[22]:
```
     symboling normalized-losses          make fuel-type aspiration  \
0            3                 ?  alfa-romero       gas        std
```

```
1            3                 ?    alfa-romero       gas        std
72           3               142  mercedes-benz       gas        std
128          3                 ?        porsche       gas        std
172          2               134         toyota       gas        std
189          3                 ?     volkswagen       gas        std

    num-of-doors   body-style drive-wheels engine-location  wheel-base  …  \
0            two  convertible          rwd           front        88.6  …
1            two  convertible          rwd           front        88.6  …
72           two  convertible          rwd           front        96.6  …
128          two  convertible          rwd            rear        89.5  …
172          two  convertible          rwd           front        98.4  …
189          two  convertible          fwd           front        94.5  …

     engine-size  fuel-system  bore  stroke compression-ratio horsepower  \
0            130         mpfi  3.47    2.68               9.0        111
1            130         mpfi  3.47    2.68               9.0        111
72           234         mpfi  3.46     3.1               8.3        155
128          194         mpfi  3.74     2.9               9.5        207
172          146         mpfi  3.62     3.5               9.3        116
189          109         mpfi  3.19     3.4               8.5         90

     peak-rpm city-mpg highway-mpg  price
0        5000       21          27  13495
1        5000       21          27  16500
72       4750       16          18  35056
128      5900       17          25  37028
172      4800       24          30  17669
189      5500       24          29  11595

[6 rows x 26 columns]
```

[23]:
```
#other ways to write the same quesry as aove
style = df.groupby('body-style')
#To print the values contained in group convertible
style.get_group("convertible")
```

[23]:
```
     symboling normalized-losses          make fuel-type aspiration  \
0            3                 ?    alfa-romero       gas        std
1            3                 ?    alfa-romero       gas        std
72           3               142  mercedes-benz       gas        std
128          3                 ?        porsche       gas        std
172          2               134         toyota       gas        std
189          3                 ?     volkswagen       gas        std

    num-of-doors   body-style drive-wheels engine-location  wheel-base  …  \
0            two  convertible          rwd           front        88.6  …
```

```
1            two  convertible          rwd           front        88.6  …
72           two  convertible          rwd           front        96.6  …
128          two  convertible          rwd            rear        89.5  …
172          two  convertible          rwd           front        98.4  …
189          two  convertible          fwd           front        94.5  …

     engine-size  fuel-system  bore  stroke compression-ratio horsepower  \
0            130         mpfi  3.47    2.68               9.0        111
1            130         mpfi  3.47    2.68               9.0        111
72           234         mpfi  3.46     3.1               8.3        155
128          194         mpfi  3.74     2.9               9.5        207
172          146         mpfi  3.62     3.5               9.3        116
189          109         mpfi  3.19     3.4               8.5         90

     peak-rpm city-mpg highway-mpg  price
0        5000       21          27  13495
1        5000       21          27  16500
72       4750       16          18  35056
128      5900       17          25  37028
172      4800       24          30  17669
189      5500       24          29  11595

[6 rows x 26 columns]
```

1.1 Selecting a subset of columns

To form groups based on multiple categories, we can simply specify the column names in the groupby() function. Grouping will be done simultaneously with the first category, the second category, and so on.

[145]:
```
double_grouping = df.groupby(["body-style","drive-wheels"])
#To print the first values contained in each group
double_grouping.first()
```

[145]:
```
                          symboling normalized-losses  … highway-mpg  price
body-style  drive-wheels                               …
convertible fwd                   3                 ?  …          29  11595
            rwd                   3                 ?  …          27  13495
hardtop     fwd                   2               168  …          37   8249
            rwd                   0                93  …          25  28176
hatchback   4wd                   0                 ?  …          22      ?
            fwd                   2               121  …          53   5151
            rwd                   1                 ?  …          26  16500
sedan       4wd                   2               164  …          22  17450
            fwd                   2               164  …          30  13950
            rwd                   2               192  …          29  16430
wagon       4wd                   0                85  …          29   8013
```

```
fwd                  1                 ?  …          25  18920
rwd                 -1                93  …          25  28248

[13 rows x 24 columns]
```

1.1.1 Max and min

There are functions such as max(), min(), mean(), first(), and last() that can be directly applied to the GroupBy object in order to obtain summary statistics for each group.

[25]:
```
# max() will print the maximum entry of each group
df.groupby('body-style')['normalized-losses'].max()
# min() will print the minimum entry of each group
df.groupby('body-style')['normalized-losses'].min()
```

[25]:
```
body-style
convertible    134
hardtop        134
hatchback      101
sedan          102
wagon          103
Name: normalized-losses, dtype: object
```

[26]:
```
# style['normalized-losses'].min() will print output in series
df.groupby('body-style')['normalized-losses'].min()

# style[['normalized-losses']].min() will print output in data frame
df.groupby('body-style')[['normalized-losses']].min()
```

[26]:
```
             normalized-losses
body-style
convertible                134
hardtop                    134
hatchback                  101
sedan                      102
wagon                      103
```

1.1.2 Mean

We can find the mean values for the numerical column in each group. This can be done using the df.mean() method.

[28]:
```
# mean() will print mean of numerical column in each group
df.groupby('body-style').mean()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:1942, in GroupBy._agg_py_fallback(self, how, values, ndim, alt)
   1941 try:
-> 1942     res_values = self._grouper.agg_series(ser, alt, preserve_dtype=True
   1943 except Exception as err:

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪ops.py:864, in BaseGrouper.agg_series(self, obj, func, preserve_dtype)
    862     preserve_dtype = True
--> 864 result = self._aggregate_series_pure_python(obj, func)
    866 npvalues = lib.maybe_convert_objects(result, try_float=False)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪ops.py:885, in BaseGrouper._aggregate_series_pure_python(self, obj, func)
    884 for i, group in enumerate(splitter):
--> 885     res = func(group)
    886     res = extract_result(res)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:2454, in GroupBy.mean.<locals>.<lambda>(x)
   2451 else:
   2452     result = self._cython_agg_general(
   2453         "mean",
-> 2454         alt=lambda x: Series(x, copy=False).
 ↪mean(numeric_only=numeric_only),
   2455         numeric_only=numeric_only,
   2456     )
   2457     return result.__finalize__(self.obj, method="groupby")

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/series.py
 ↪6549, in Series.mean(self, axis, skipna, numeric_only, **kwargs)
   6541 @doc(make_doc("mean", ndim=1))
   6542 def mean(
   6543     self,
   (…)
   6547     **kwargs,
   6548 ):
-> 6549     return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/generic.p
 ↪12420, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  12413 def mean(
  12414     self,
  12415     axis: Axis | None = 0,
   (…)
  12418     **kwargs,
  12419 ) -> Series | float:
> 12420     return self._stat_function(
```

```
  12421         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  12422     )

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/generic.py:
 ↪12377, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only␣
 ↪**kwargs)
  12375 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12377 return self._reduce(
  12378     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_onl
  12379 )

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/series.py
 ↪6457, in Series._reduce(self, op, name, axis, skipna, numeric_only,␣
 ↪filter_type, **kwds)
   6453     raise TypeError(
   6454         f"Series.{name} does not allow {kwd_name}={numeric_only} "
   6455         "with non-numeric dtypes."
   6456     )
-> 6457 return op(delegate, skipna=skipna, **kwds)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪147, in bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪404, in _datetimelike_compat.<locals>.new_func(values, axis, skipna, mask,␣
 ↪**kwargs)
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    406 if datetimelike:

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪720, in nanmean(values, axis, skipna, mask)
    719 the_sum = values.sum(axis, dtype=dtype_sum)
--> 720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪1701, in _ensure_numeric(x)
   1699 if isinstance(x, str):
   1700     # GH#44008, GH#36703 avoid casting e.g. strings to numeric
-> 1701     raise TypeError(f"Could not convert string '{x}' to numeric")
   1702 try:

TypeError: Could not convert string '??142?134?' to numeric
```

```
The above exception was the direct cause of the following exception:

TypeError                                 Traceback (most recent call last)
Cell In[28], line 2
      1 # mean() will print mean of numerical column in each group
----> 2 df.groupby('body-style').mean()

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:2452, in GroupBy.mean(self, numeric_only, engine, engine_kwargs)
   2445     return self._numba_agg_general(
   2446         grouped_mean,
   2447         executor.float_dtype_mapping,
   2448         engine_kwargs,
   2449         min_periods=0,
   2450     )
   2451 else:
-> 2452     result = self._cython_agg_general(
   2453         "mean",
   2454         alt=lambda x: Series(x, copy=False).
 ↪mean(numeric_only=numeric_only),
   2455         numeric_only=numeric_only,
   2456     )
   2457     return result.__finalize__(self.obj, method="groupby")

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:1998, in GroupBy._cython_agg_general(self, how, alt, numeric_only,
 ↪min_count, **kwargs)
   1995     result = self._agg_py_fallback(how, values, ndim=data.ndim, alt=alt
   1996     return result
-> 1998 new_mgr = data.grouped_reduce(array_func)
   1999 res = self._wrap_agged_manager(new_mgr)
   2000 if how in ["idxmin", "idxmax"]:

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/internals
 ↪managers.py:1469, in BlockManager.grouped_reduce(self, func)
   1465 if blk.is_object:
   1466     # split on object-dtype blocks bc some columns may raise
   1467     #  while others do not.
   1468     for sb in blk._split():
-> 1469         applied = sb.apply(func)
   1470         result_blocks = extend_blocks(applied, result_blocks)
   1471 else:

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/internals
 ↪blocks.py:393, in Block.apply(self, func, **kwargs)
    387 @final
    388 def apply(self, func, **kwargs) -> list[Block]:
    389     """
```

```
    390     apply the function to my values; return a block if we are not
    391     one
    392     """
--> 393     result = func(self.values, **kwargs)
    395     result = maybe_coerce_values(result)
    396     return self._split_op_result(result)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:1995, in GroupBy._cython_agg_general.<locals>.array_func(values)
   1992     return result
   1994 assert alt is not None
-> 1995 result = self._agg_py_fallback(how, values, ndim=data.ndim, alt=alt)
   1996 return result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:1946, in GroupBy._agg_py_fallback(self, how, values, ndim, alt)
   1944     msg = f"agg function failed [how->{how},dtype->{ser.dtype}]"
   1945     # preserve the kind of exception that raised
-> 1946     raise type(err)(msg) from err
   1948 if ser.dtype == object:
   1949     res_values = res_values.astype(object, copy=False)

TypeError: agg function failed [how->mean,dtype->object]
```

[29]:
```
# get mean of each column of specific group
style.get_group("convertible").mean()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 2
      1 # get mean of each column of specific group
----> 2 style.get_group("convertible").mean()

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/frame.py:
 ↪11693, in DataFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  11685 @doc(make_doc("mean", ndim=2))
  11686 def mean(
  11687     self,
   (…)
  11691     **kwargs,
  11692 ):
> 11693     result = super().mean(axis, skipna, numeric_only, **kwargs)
  11694     if isinstance(result, Series):
  11695         result = result.__finalize__(self, method="mean")

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/generic.p
 ↪12420, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
```

```
  12413 def mean(
  12414     self,
  12415     axis: Axis | None = 0,
   (…)
  12418     **kwargs,
  12419 ) -> Series | float:
> 12420     return self._stat_function(
  12421         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  12422     )

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/generic.py:
 ↪12377, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only␣
 ↪**kwargs)
  12373 nv.validate_func(name, (), kwargs)
  12375 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12377 return self._reduce(
  12378     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_onl
  12379 )

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/frame.py:
 ↪11562, in DataFrame._reduce(self, op, name, axis, skipna, numeric_only,␣
 ↪filter_type, **kwds)
  11558     df = df.T
  11560 # After possibly _get_data and transposing, we are now in the
  11561 #  simple case where we can use BlockManager.reduce
> 11562 res = df._mgr.reduce(blk_func)
  11563 out = df._constructor_from_mgr(res, axes=res.axes).iloc[0]
  11564 if out_dtype is not None and out.dtype != "boolean":

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/internals/
 ↪managers.py:1500, in BlockManager.reduce(self, func)
   1498 res_blocks: list[Block] = []
   1499 for blk in self.blocks:
-> 1500     nbs = blk.reduce(func)
   1501     res_blocks.extend(nbs)
   1503 index = Index([None])  # placeholder

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/internals/
 ↪blocks.py:404, in Block.reduce(self, func)
    398 @final
    399 def reduce(self, func) -> list[Block]:
    400     # We will apply the function and reshape the result into a single-r w
    401     #  Block with the same mgr_locs; squeezing will be done at a higher
 ↪level
    402     assert self.ndim == 2
--> 404     result = func(self.values)
    406     if self.values.ndim == 1:
    407         res_values = result
```

```
File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/frame.py:
 ↪11481, in DataFrame._reduce.<locals>.blk_func(values, axis)
  11479         return np.array([result])
  11480 else:
> 11481     return op(values, axis=axis, skipna=skipna, **kwds)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪147, in bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    145         result = alt(values, axis=axis, skipna=skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪404, in _datetimelike_compat.<locals>.new_func(values, axis, skipna, mask,␣
 ↪**kwargs)
    401 if datetimelike and mask is None:
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    406 if datetimelike:
    407     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪720, in nanmean(values, axis, skipna, mask)
    718 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
    719 the_sum = values.sum(axis, dtype=dtype_sum)
--> 720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):
    723     count = cast(np.ndarray, count)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪1686, in _ensure_numeric(x)
   1683 inferred = lib.infer_dtype(x)
   1684 if inferred in ["string", "mixed"]:
   1685     # GH#44008, GH#36703 avoid casting e.g. strings to numeric
-> 1686     raise TypeError(f"Could not convert {x} to numeric")
   1687 try:
   1688     x = x.astype(np.complex128)

TypeError: Could not convert ['??142?134?'␣
 ↪'alfa-romeroalfa-romeromercedes-benzporschetoyotavolkswagen'
 'gasgasgasgasgasgas' 'stdstdstdstdstdstd' 'twotwotwotwotwotwo'
 'convertibleconvertibleconvertibleconvertibleconvertibleconvertible'
 'rwdrwdrwdrwdrwdfwd' 'frontfrontfrontrearfrontfront'
 'dohcdohcohcvohcfohcohc' 'fourfoureightsixfourfour'
 'mpfimpfimpfimpfimpfimpfi' '3.473.473.463.743.623.19'
 '2.682.683.12.93.53.4' '11111115520711690' '5000500047505900480055000'
```

```
'134951650035056370281766911595'] to numeric
```

1.1.3 Data aggregation

Aggregation is the process of implementing any mathematical operation on a dataset or a subset of it. Aggregation is one of the many techniques in pandas that's used to manipulate the data in the dataframe for data analysis. The Dataframe.aggregate() function is used to apply aggregation across one or more columns. Some of the most frequently used aggregations are as follows:

1. sum: Returns the sum of the values for the requested axis
2. min: Returns the minimum of the values for the requested axis
3. max: Returns the maximum of the values for the requested axis

We can apply aggregation in a DataFrame, df, as df.aggregate() or df.agg().

Aggregation only works with numeric type columns

[31]:
```
# get the sum of the price for each body-style group
df.groupby('body-style')['price'].sum()
```

[31]:
```
body-style
convertible                          134951650035056370281766911595
hardtop                     28176454008249325283402884499639111 99
hatchback      16500?5151629555726377795762291296464796855539…
sedan          139501745015250177102387516430169252097021105 2…
wagon          189208921729528248734979991439912440138601669 5…
Name: price, dtype: object
```

[32]:
```
# get the number of symboling/records in each group
style['symboling'].count()
```

[32]:
```
body-style
convertible     6
hardtop         8
hatchback      70
sedan          96
wagon          25
Name: symboling, dtype: int64
```

[34]:
```
# new dataframe that consist length,width,height,curb-weight and price
new_dataset = df.
  ↪filter(["length","width","height","curb-weight","price"],axis=1)
new_dataset
```

[34]:

	length	width	height	curb-weight	price
0	168.8	64.1	48.8	2548	13495
1	168.8	64.1	48.8	2548	16500
2	171.2	65.5	52.4	2823	16500

```
3     176.6   66.2    54.3          2337  13950
4     176.6   66.4    54.3          2824  17450
..      …     …       …      …      …
200   188.8   68.9    55.5          2952  16845
201   188.8   68.8    55.5          3049  19045
202   188.8   68.9    55.5          3012  21485
203   188.8   68.9    55.5          3217  22470
204   188.8   68.9    55.5          3062  22625

[205 rows x 5 columns]
```

[35]:
```
# applying single aggregation for mean over the columns
new_dataset.agg("mean", axis="rows")# applying aggregation sum and minimum␣
 ↪across all the columns
new_dataset.agg(['sum', 'min'])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[35], line 2
      1 # applying single aggregation for mean over the columns
----> 2 new_dataset.agg("mean", axis="rows")

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/frame.py:
 ↪10149, in DataFrame.aggregate(self, func, axis, *args, **kwargs)
  10146 axis = self._get_axis_number(axis)
  10148 op = frame_apply(self, func=func, axis=axis, args=args, kwargs=kwargs)
> 10149 result = op.agg()
  10150 result = reconstruct_and_relabel_result(result, func, **kwargs)
  10151 return result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪928, in FrameApply.agg(self)
    926 result = None
    927 try:
--> 928     result = super().agg()
    929 finally:
    930     self.obj = obj

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪187, in Apply.agg(self)
    184 kwargs = self.kwargs
    186 if isinstance(func, str):
--> 187     return self.apply_str()
    189 if is_dict_like(func):
    190     return self.agg_dict_like()
```

```
File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪1131, in FrameApply.apply_str(self)
   1129     value = obj.shape[self.axis]
   1130     return obj._constructor_sliced(value, index=self.agg_axis)
-> 1131 return super().apply_str()

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪603, in Apply.apply_str(self)
    601         else:
    602             self.kwargs["axis"] = self.axis
--> 603 return self._apply_str(obj, func, *self.args, **self.kwargs)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪693, in Apply._apply_str(self, obj, func, *args, **kwargs)
    691 f = getattr(obj, func)
    692 if callable(f):
--> 693     return f(*args, **kwargs)
    695 # people may aggregate on a non-callable attribute
    696 # but don't let them think they can pass args to it
    697 assert len(args) == 0

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/frame.py:
 ↪11693, in DataFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  11685 @doc(make_doc("mean", ndim=2))
  11686 def mean(
  11687     self,
   (…)
  11691     **kwargs,
  11692 ):
> 11693     result = super().mean(axis, skipna, numeric_only, **kwargs)
  11694     if isinstance(result, Series):
  11695         result = result.__finalize__(self, method="mean")

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/generic.py:
 ↪12420, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  12413 def mean(
  12414     self,
  12415     axis: Axis | None = 0,
   (…)
  12418     **kwargs,
  12419 ) -> Series | float:
> 12420     return self._stat_function(
  12421         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  12422     )

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/generic.py:
 ↪12377, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only␣
 ↪**kwargs)
```

```
  12373 nv.validate_func(name, (), kwargs)
  12375 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12377 return self._reduce(
  12378     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_onl
  12379 )

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/frame.py:
 ↪11562, in DataFrame._reduce(self, op, name, axis, skipna, numeric_only,␣
 ↪filter_type, **kwds)
  11558     df = df.T
  11560 # After possibly _get_data and transposing, we are now in the
  11561 #  simple case where we can use BlockManager.reduce
> 11562 res = df._mgr.reduce(blk_func)
  11563 out = df._constructor_from_mgr(res, axes=res.axes).iloc[0]
  11564 if out_dtype is not None and out.dtype != "boolean":

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/internals
 ↪managers.py:1500, in BlockManager.reduce(self, func)
   1498 res_blocks: list[Block] = []
   1499 for blk in self.blocks:
-> 1500     nbs = blk.reduce(func)
   1501     res_blocks.extend(nbs)
   1503 index = Index([None])  # placeholder

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/internals
 ↪blocks.py:404, in Block.reduce(self, func)
    398 @final
    399 def reduce(self, func) -> list[Block]:
    400     # We will apply the function and reshape the result into a single-r w
    401     #  Block with the same mgr_locs; squeezing will be done at a higher
 ↪level
    402     assert self.ndim == 2
--> 404     result = func(self.values)
    406     if self.values.ndim == 1:
    407         res_values = result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/frame.py:
 ↪11481, in DataFrame._reduce.<locals>.blk_func(values, axis)
  11479         return np.array([result])
  11480 else:
> 11481     return op(values, axis=axis, skipna=skipna, **kwds)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪147, in bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    145         result = alt(values, axis=axis, skipna=skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result
```

```
File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪404, in _datetimelike_compat.<locals>.new_func(values, axis, skipna, mask,␣
 ↪**kwargs)
    401 if datetimelike and mask is None:
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    406 if datetimelike:
    407     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪720, in nanmean(values, axis, skipna, mask)
    718 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
    719 the_sum = values.sum(axis, dtype=dtype_sum)
--> 720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):
    723     count = cast(np.ndarray, count)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪1686, in _ensure_numeric(x)
   1683 inferred = lib.infer_dtype(x)
   1684 if inferred in ["string", "mixed"]:
   1685     # GH#44008, GH#36703 avoid casting e.g. strings to numeric
-> 1686     raise TypeError(f"Could not convert {x} to numeric")
   1687 try:
   1688     x = x.astype(np.complex128)

TypeError: Could not convert ['1349516500165001395017450152501771018920238752
 ↪1643016925209702110524565307604131536880515162956575557263777957622966927609
 ↪?
 ↪1104832250355503600051956095679566957395109451184513645156458845849510595102
 ↪9295989511850121701504015510181501862051187053760371267775996092331125974631
 ↪to numeric
```

[36]:
```
# applying aggregation sum and minimum across all the columns
new_dataset.agg(['sum', 'min'])
```

[36]:

	length	width	height	curb-weight	\
sum	35680.1	13511.1	11013.6	523891	
min	141.1	60.3	47.8	1488	

	price
sum	134951650016500139501745015250177101892023875?…
min	10198

[37]:
```
# To apply aggregation functions across different columns, you can pass a␣
 ↪dictionary with a key containing the column names and values containing the␣
 ↪list of aggregation functions for any specific column:
# find aggregation for these columns
```

```
new_dataset.aggregate({"length":['sum', 'min'],
"width":['max', 'min'],
"height":['min', 'sum'],
"curb-weight":['sum']})
# if any specific aggregation is not applied on a column
# then it has NaN value corresponding to it
```

[37]:

	length	width	height	curb-weight
sum	35680.1	NaN	11013.6	523891.0
min	141.1	60.3	47.8	NaN
max	NaN	72.3	NaN	NaN

[ ]:

1.1.4 Group-wise operations

The most important operations groupBy implements are aggregate, filter, transform, and apply. An efficient way of implementing aggregation functions in the dataset is by doing so after grouping the required columns. The aggregated function will return a single aggregated value for each group. Once these groups have been created, we can apply several aggregation operations to that grouped data.

[40]:
```
# Group the data frame df by body-style and drive-wheels and extract stats from␣
 ↪each group
df.groupby(
    ["body-style","drive-wheels"]
    ).agg(
    {
    'height':min, # minimum height of car in each group
    'length': max, # maximum length of car in each group
    'price': 'mean', # average price of car in each group
    }
)
```

```
/tmp/ipykernel_156873/3089003666.py:4: FutureWarning: The provided callable
<built-in function min> is currently using SeriesGroupBy.min. In a future
version of pandas, the provided callable will be used directly. To keep current
behavior pass the string "min" instead.
  ).agg(
/tmp/ipykernel_156873/3089003666.py:4: FutureWarning: The provided callable
<built-in function max> is currently using SeriesGroupBy.max. In a future
version of pandas, the provided callable will be used directly. To keep current
behavior pass the string "max" instead.
  ).agg(
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:1942, in GroupBy._agg_py_fallback(self, how, values, ndim, alt)
   1941 try:
-> 1942     res_values = self._grouper.agg_series(ser, alt, preserve_dtype=True
   1943 except Exception as err:

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪ops.py:864, in BaseGrouper.agg_series(self, obj, func, preserve_dtype)
    862     preserve_dtype = True
--> 864 result = self._aggregate_series_pure_python(obj, func)
    866 npvalues = lib.maybe_convert_objects(result, try_float=False)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪ops.py:885, in BaseGrouper._aggregate_series_pure_python(self, obj, func)
    884 for i, group in enumerate(splitter):
--> 885     res = func(group)
    886     res = extract_result(res)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:2454, in GroupBy.mean.<locals>.<lambda>(x)
   2451 else:
   2452     result = self._cython_agg_general(
   2453         "mean",
-> 2454         alt=lambda x: Series(x, copy=False).
 ↪mean(numeric_only=numeric_only),
   2455         numeric_only=numeric_only,
   2456     )
   2457     return result.__finalize__(self.obj, method="groupby")

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/series.py
 ↪6549, in Series.mean(self, axis, skipna, numeric_only, **kwargs)
   6541 @doc(make_doc("mean", ndim=1))
   6542 def mean(
   6543     self,
   (…)
   6547     **kwargs,
   6548 ):
-> 6549     return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/generic.p
 ↪12420, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  12413 def mean(
  12414     self,
  12415     axis: Axis | None = 0,
   (…)
  12418     **kwargs,
  12419 ) -> Series | float:
> 12420     return self._stat_function(
```

```
  12421         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  12422     )

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/generic.py:
 ↪12377, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only␣
 ↪**kwargs)
  12375 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12377 return self._reduce(
  12378     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_onl
  12379 )

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/series.py
 ↪6457, in Series._reduce(self, op, name, axis, skipna, numeric_only,␣
 ↪filter_type, **kwds)
   6453     raise TypeError(
   6454         f"Series.{name} does not allow {kwd_name}={numeric_only} "
   6455         "with non-numeric dtypes."
   6456     )
-> 6457 return op(delegate, skipna=skipna, **kwds)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪147, in bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪404, in _datetimelike_compat.<locals>.new_func(values, axis, skipna, mask,␣
 ↪**kwargs)
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    406 if datetimelike:

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪720, in nanmean(values, axis, skipna, mask)
    719 the_sum = values.sum(axis, dtype=dtype_sum)
--> 720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/nanops.py
 ↪1701, in _ensure_numeric(x)
   1699 if isinstance(x, str):
   1700     # GH#44008, GH#36703 avoid casting e.g. strings to numeric
-> 1701     raise TypeError(f"Could not convert string '{x}' to numeric")
   1702 try:

TypeError: Could not convert string '11595' to numeric
```

```
The above exception was the direct cause of the following exception:

TypeError                                 Traceback (most recent call last)
Cell In[40], line 4
      1 # Group the data frame df by body-style and drive-wheels and extract␣
 ↪stats from each group
      2 df.groupby(
      3     ["body-style","drive-wheels"]
----> 4     ).agg(
      5     {
      6     'height':min, # minimum height of car in each group
      7     'length': max, # maximum length of car in each group
      8     'price': 'mean', # average price of car in each group
      9     }
     10 )

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪generic.py:1432, in DataFrameGroupBy.aggregate(self, func, engine,␣
 ↪engine_kwargs, *args, **kwargs)
   1429     kwargs["engine_kwargs"] = engine_kwargs
   1431 op = GroupByApply(self, func, args=args, kwargs=kwargs)
-> 1432 result = op.agg()
   1433 if not is_dict_like(func) and result is not None:
   1434     # GH #52849
   1435     if not self.as_index and is_list_like(func):

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪190, in Apply.agg(self)
    187     return self.apply_str()
    189 if is_dict_like(func):
--> 190     return self.agg_dict_like()
    191 elif is_list_like(func):
    192     # we require a list, but not a 'str'
    193     return self.agg_list_like()

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪423, in Apply.agg_dict_like(self)
    415 def agg_dict_like(self) -> DataFrame | Series:
    416     """
    417     Compute aggregation in the case of a dict-like argument.
    418
   (…)
    421     Result of aggregation.
    422     """
--> 423     return self.agg_or_apply_dict_like(op_name="agg")

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪1608, in GroupByApply.agg_or_apply_dict_like(self, op_name)
```

```
   1603     kwargs.update({"engine": engine, "engine_kwargs": engine_kwargs})
   1605 with com.temp_setattr(
   1606     obj, "as_index", True, condition=hasattr(obj, "as_index")
   1607 ):
-> 1608     result_index, result_data = self.compute_dict_like(
   1609         op_name, selected_obj, selection, kwargs
   1610     )
   1611 result = self.wrap_results_dict_like(selected_obj, result_index,␣
 ↪result_data)
   1612 return result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪497, in Apply.compute_dict_like(self, op_name, selected_obj, selection, kwarg
    493         results += key_data
    494 else:
    495     # key used for column selection and output
    496     results = [
--> 497         getattr(obj._gotitem(key, ndim=1), op_name)(how, **kwargs)
    498         for key, how in func.items()
    499     ]
    500     keys = list(func.keys())
    502 return keys, results

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪generic.py:249, in SeriesGroupBy.aggregate(self, func, engine, engine_kwargs,
 ↪*args, **kwargs)
    247     if engine_kwargs is not None:
    248         kwargs["engine_kwargs"] = engine_kwargs
--> 249     return getattr(self, func)(*args, **kwargs)
    251 elif isinstance(func, abc.Iterable):
    252     # Catch instances of lists / tuples
    253     # but not the class list / tuple itself.
    254     func = maybe_mangle_lambdas(func)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:2452, in GroupBy.mean(self, numeric_only, engine, engine_kwargs)
   2445     return self._numba_agg_general(
   2446         grouped_mean,
   2447         executor.float_dtype_mapping,
   2448         engine_kwargs,
   2449         min_periods=0,
   2450     )
   2451 else:
-> 2452     result = self._cython_agg_general(
   2453         "mean",
   2454         alt=lambda x: Series(x, copy=False).
 ↪mean(numeric_only=numeric_only),
   2455         numeric_only=numeric_only,
```

```
   2456     )
   2457     return result.__finalize__(self.obj, method="groupby")

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:1998, in GroupBy._cython_agg_general(self, how, alt, numeric_only,
 ↪min_count, **kwargs)
   1995     result = self._agg_py_fallback(how, values, ndim=data.ndim, alt=alt
   1996     return result
-> 1998 new_mgr = data.grouped_reduce(array_func)
   1999 res = self._wrap_agged_manager(new_mgr)
   2000 if how in ["idxmin", "idxmax"]:

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/internals
 ↪base.py:367, in SingleDataManager.grouped_reduce(self, func)
    365 def grouped_reduce(self, func):
    366     arr = self.array
--> 367     res = func(arr)
    368     index = default_index(len(res))
    370     mgr = type(self).from_array(res, index)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:1995, in GroupBy._cython_agg_general.<locals>.array_func(values)
   1992     return result
   1994 assert alt is not None
-> 1995 result = self._agg_py_fallback(how, values, ndim=data.ndim, alt=alt)
   1996 return result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/groupby/
 ↪groupby.py:1946, in GroupBy._agg_py_fallback(self, how, values, ndim, alt)
   1944     msg = f"agg function failed [how->{how},dtype->{ser.dtype}]"
   1945     # preserve the kind of exception that raised
-> 1946     raise type(err)(msg) from err
   1948 if ser.dtype == object:
   1949     res_values = res_values.astype(object, copy=False)

TypeError: agg function failed [how->mean,dtype->object]
```

[ ]:

2 Merging on Index

[ ]:
```
from IPython.display import display_html

def display_side_by_side(*args):
    html_str=''
    for df in args:
```

```
        html_str+=df.to_html()
    display_html(html_str.replace('table','table style="display:
 ↪inline"'),raw=True)
```

[153]:
```
import pandas as pd

left1 = pd.DataFrame({'key': ['apple','ball','apple', 'apple', 'ball', 'cat'],␣
 ↪'value': range(6)})
right1 = pd.DataFrame({'group_val': [33.4, 5]}, index=['apple', 'ball'])
display_side_by_side(left1,right1)

df1 = pd.merge(left1, right1, left_on='key', right_index=True)
df1
```

[153]:

	key	value	group_val
0	apple	0	33.4
2	apple	2	33.4
3	apple	3	33.4
1	ball	1	5.0
4	ball	4	5.0

[154]:
```
df2 = pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
df2
```

[154]:

	key	value	group_val
0	apple	0	33.4
2	apple	2	33.4
3	apple	3	33.4
1	ball	1	5.0
4	ball	4	5.0
5	cat	5	NaN

3 Concatenating along an axis

[155]:
```
import numpy as np

yarra = np.arange(15).reshape((3,5))
yarra
```

[155]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

[156]:
```
 np.concatenate([yarra, yarra], axis=1)
```

[156]:
```
array([[ 0,  1,  2,  3,  4,  0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 10, 11, 12, 13, 14]])
```

Data Aggregation

[157]:
```
# new dataframe that consist length,width,height,curb-weight and price
new_dataset = df.
 ↪filter(["length","width","height","curb-weight","price"],axis=1)
new_dataset
```

[157]:

	length	width	height	curb-weight	price
0	168.8	64.1	48.8	2548	13495
1	168.8	64.1	48.8	2548	16500
2	171.2	65.5	52.4	2823	16500
3	176.6	66.2	54.3	2337	13950
4	176.6	66.4	54.3	2824	17450
..	…	…	…	…	…
200	188.8	68.9	55.5	2952	16845
201	188.8	68.8	55.5	3049	19045
202	188.8	68.9	55.5	3012	21485
203	188.8	68.9	55.5	3217	22470
204	188.8	68.9	55.5	3062	22625

[205 rows x 5 columns]

[158]:
```
# applying single aggregation for mean over the columns
new_dataset.agg("mean", axis="rows")
```

[158]:
```
length          174.049268
width            65.907805
height           53.724878
curb-weight    2555.565854
dtype: float64
```

[159]:
```
# applying aggregation sum and minimun across all the columns
new_dataset.agg(['sum', 'min'])
```

[159]:

	length	…	price
sum	35680.1	…	134951650016500139501745015250177101892023875?…
min	141.1	…	10198

[2 rows x 5 columns]

[160]:
```
# find aggregation for these columns
new_dataset.aggregate({"length":['sum', 'min'],
              "width":['max', 'min'],
```

```
            "height":['min', 'sum'],
            "curb-weight":['sum']})
# if any specific aggregation is not applied on a column
# then it has NaN value corresponding to it
```

[160]:

	length	width	height	curb-weight
max	NaN	72.3	NaN	NaN
min	141.1	60.3	47.8	NaN
sum	35680.1	NaN	11013.6	523891.0

[161]:
```
df.dtypes
```

[161]:
```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                 object
dtype: object
```

[162]:
```
df['price'].str.isnumeric().value_counts()
df['price'].loc[df['price'].str.isnumeric() == False]
price = df['price'].loc[df['price'] != '?']
pmean = price.astype(str).astype(int).mean()
df['price'] = df['price'].replace('?',pmean).astype(int)
```

```
df.groupby(["body-style","drive-wheels"]).agg({'height': min, 'length': max,␣
 ↪'price': 'mean'})
```

[162]:

body-style	drive-wheels	height	length	price
convertible	fwd	55.6	159.3	11595.000000
	rwd	48.8	180.3	23949.600000
hardtop	fwd	53.3	162.4	8249.000000
	rwd	51.6	199.2	24202.714286
hatchback	4wd	52.0	178.2	10405.000000
	fwd	49.4	186.6	8396.387755
	rwd	49.6	183.5	14278.263158
sedan	4wd	54.3	176.6	12647.333333
	fwd	50.6	192.7	9930.929825
	rwd	47.8	208.1	21711.833333
wagon	4wd	54.9	173.6	9095.750000
	fwd	53.0	192.7	9997.333333
	rwd	54.1	198.9	16994.222222

Group-wise operations

[163]:
```
# Group the data frame df by body-style and drive-wheels and extract stats from␣
 ↪each group
df.groupby(
   ["body-style","drive-wheels"]
).agg(
    {
         'height': min,    # minimum height of car in each group
         'length': max,  # maximum length of car in each group
         'price': 'mean',  # average price of car in each group

    }
)
```

[163]:

body-style	drive-wheels	height	length	price
convertible	fwd	55.6	159.3	11595.000000
	rwd	48.8	180.3	23949.600000
hardtop	fwd	53.3	162.4	8249.000000
	rwd	51.6	199.2	24202.714286
hatchback	4wd	52.0	178.2	10405.000000
	fwd	49.4	186.6	8396.387755
	rwd	49.6	183.5	14278.263158
sedan	4wd	54.3	176.6	12647.333333
	fwd	50.6	192.7	9930.929825
	rwd	47.8	208.1	21711.833333
wagon	4wd	54.9	173.6	9095.750000

```
                fwd             53.0   192.7   9997.333333
                rwd             54.1   198.9  16994.222222
```

[164]:
```
# create dictionary of aggregrations
aggregations=(
    {
          'height':min,     # minimum height of car in each group
          'length': max,  # maximum length of car in each group
          'price': 'mean',  # average price of car in each group

    }
)
# implementing aggregations in groups
df.groupby(
   ["body-style","drive-wheels"]
).agg(aggregations)
```

[164]:
```
                          height  length         price
body-style  drive-wheels
convertible fwd             55.6   159.3  11595.000000
            rwd             48.8   180.3  23949.600000
hardtop     fwd             53.3   162.4   8249.000000
            rwd             51.6   199.2  24202.714286
hatchback   4wd             52.0   178.2  10405.000000
            fwd             49.4   186.6   8396.387755
            rwd             49.6   183.5  14278.263158
sedan       4wd             54.3   176.6  12647.333333
            fwd             50.6   192.7   9930.929825
            rwd             47.8   208.1  21711.833333
wagon       4wd             54.9   173.6   9095.750000
            fwd             53.0   192.7   9997.333333
            rwd             54.1   198.9  16994.222222
```

We can use numpy functions in aggregation as well

[165]:
```
# import the numpy library as np
import numpy as np
# using numpy libraries for operations
df.groupby(
   ["body-style","drive-wheels"])["price"].agg([np.sum, np.mean, np.std])
```

[165]:
```
                            sum          mean           std
body-style  drive-wheels
convertible fwd           11595  11595.000000           NaN
            rwd          119748  23949.600000  11165.099700
hardtop     fwd            8249   8249.000000           NaN
            rwd          169419  24202.714286  14493.311190
```

```
hatchback   4wd            20810  10405.000000   3962.626402
            fwd           411423   8396.387755   3004.675695
            rwd           271287  14278.263158   3732.860727
sedan       4wd            37942  12647.333333   4280.814681
            fwd           566063   9930.929825   3513.098067
            rwd           781626  21711.833333   9194.820239
wagon       4wd            36383   9095.750000   1775.652063
            fwd           119968   9997.333333   3584.185551
            rwd           152948  16994.222222   4686.703313
```

Renaming grouped aggregation columns

[166]:
```
df.groupby(
   ["body-style","drive-wheels"]).agg(
    # Get max of the price column for each group
    max_price=('price', max),
    # Get min of the price column for each group
    min_price=('price', min),
    # Get sum of the price column for each group
    total_price=('price', 'mean')

)
```

[166]:

body-style	drive-wheels	max_price	min_price	total_price
convertible	fwd	11595	11595	11595.000000
	rwd	37028	13495	23949.600000
hardtop	fwd	8249	8249	8249.000000
	rwd	45400	8449	24202.714286
hatchback	4wd	13207	7603	10405.000000
	fwd	18150	5118	8396.387755
	rwd	22018	8238	14278.263158
sedan	4wd	17450	9233	12647.333333
	fwd	23875	5499	9930.929825
	rwd	41315	6785	21711.833333
wagon	4wd	11694	7898	9095.750000
	fwd	18920	6918	9997.333333
	rwd	28248	12440	16994.222222

3.1 Group-wise transformations

Performing a transformation on a group or a column returns an object that is indexed by the same axis length as itself. It is an operation that's used in conjunction with groupby(). The aggregation operation has to return a reduced version of the data, whereas the transformation operation can return a transformed version of the full data.

[41]:
```
df["price"]=df["price"].transform(lambda x:x + x/10)
df.loc[:,'price']
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪314, in Apply.transform_str_or_callable(self, func)
    313 try:
--> 314     return obj.apply(func, args=args, **kwargs)
    315 except Exception:

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/series.py
 ↪4924, in Series.apply(self, func, convert_dtype, args, by_row, **kwargs)
   4798 """
   4799 Invoke function on values of Series.
   4800
   (…)
   4915 dtype: float64
   4916 """
   4917 return SeriesApply(
   4918     self,
   4919     func,
   4920     convert_dtype=convert_dtype,
   4921     by_row=by_row,
   4922     args=args,
   4923     kwargs=kwargs,
-> 4924 ).apply()

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪1427, in SeriesApply.apply(self)
   1426 # self.func is Callable
-> 1427 return self.apply_standard()

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪1507, in SeriesApply.apply_standard(self)
   1506 action = "ignore" if isinstance(obj.dtype, CategoricalDtype) else None
-> 1507 mapped = obj._map_values(
   1508     mapper=curried, na_action=action, convert=self.convert_dtype
   1509 )
   1511 if len(mapped) and isinstance(mapped[0], ABCSeries):
   1512     # GH#43986 Need to do list(mapped) in order to get treated as neste
   1513     #  See also GH#25959 regarding EA support

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/base.py:
 ↪921, in IndexOpsMixin._map_values(self, mapper, na_action, convert)
    919     return arr.map(mapper, na_action=na_action)
--> 921 return algorithms.map_array(arr, mapper, na_action=na_action,␣
 ↪convert=convert)
```

```
File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/algorithms.
 ↪py:1743, in map_array(arr, mapper, na_action, convert)
   1742 if na_action is None:
-> 1743     return lib.map_infer(values, mapper, convert=convert)
   1744 else:

File lib.pyx:2972, in pandas._libs.lib.map_infer()

Cell In[41], line 1, in <lambda>(x)
----> 1 df["price"]=df["price"].transform(lambda x:x + x/10)
      2 df.loc[:,'price']

TypeError: unsupported operand type(s) for /: 'str' and 'int'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/ops/
 ↪array_ops.py:218, in _na_arithmetic_op(left, right, op, is_cmp)
    217 try:
--> 218     result = func(left, right)
    219 except TypeError:

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/
 ↪computation/expressions.py:242, in evaluate(op, a, b, use_numexpr)
    240     if use_numexpr:
    241         # error: "None" not callable
--> 242         return _evaluate(op, op_str, a, b)  # type: ignore[misc]
    243 return _evaluate_standard(op, op_str, a, b)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/
 ↪computation/expressions.py:131, in _evaluate_numexpr(op, op_str, a, b)
    130 if result is None:
--> 131     result = _evaluate_standard(op, op_str, a, b)
    133 return result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/
 ↪computation/expressions.py:73, in _evaluate_standard(op, op_str, a, b)
     72     _store_test_result(False)
---> 73 return op(a, b)

TypeError: unsupported operand type(s) for /: 'str' and 'int'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
Cell In[41], line 1
----> 1 df["price"]=df["price"].transform(lambda x:x + x/10)
```

```
      2 df.loc[:,'price']

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/series.py
 ↪4786, in Series.transform(self, func, axis, *args, **kwargs)
   4780 self._get_axis_number(axis)
   4781 ser = (
   4782     self.copy(deep=False)
   4783     if using_copy_on_write() or warn_copy_on_write()
   4784     else self
   4785 )
-> 4786 result = SeriesApply(ser, func=func, args=args, kwargs=kwargs).
 ↪transform()
   4787 return result

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪246, in Apply.transform(self)
    244 func = cast(AggFuncTypeBase, func)
    245 try:
--> 246     result = self.transform_str_or_callable(func)
    247 except TypeError:
    248     raise

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/apply.py:
 ↪316, in Apply.transform_str_or_callable(self, func)
    314     return obj.apply(func, args=args, **kwargs)
    315 except Exception:
--> 316     return func(obj, *args, **kwargs)

Cell In[41], line 1, in <lambda>(x)
----> 1 df["price"]=df["price"].transform(lambda x:x + x/10)
      2 df.loc[:,'price']

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/ops/commo
 ↪py:76, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
     72             return NotImplemented
     74 other = item_from_zerodim(other)
---> 76 return method(self, other)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/arraylike
 ↪py:210, in OpsMixin.__truediv__(self, other)
    208 @unpack_zerodim_and_defer("__truediv__")
    209 def __truediv__(self, other):
--> 210     return self._arith_method(other, operator.truediv)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/series.py
 ↪6135, in Series._arith_method(self, other, op)
   6133 def _arith_method(self, other, op):
   6134     self, other = self._align_for_op(other)
```

```
-> 6135     return base.IndexOpsMixin._arith_method(self, other, op)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/base.py:
 ↪1382, in IndexOpsMixin._arith_method(self, other, op)
   1379     rvalues = np.arange(rvalues.start, rvalues.stop, rvalues.step)
   1381 with np.errstate(all="ignore"):
-> 1382     result = ops.arithmetic_op(lvalues, rvalues, op)
   1384 return self._construct_result(result, name=res_name)

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/ops/
 ↪array_ops.py:283, in arithmetic_op(left, right, op)
    279     _bool_arith_check(op, left, right)  # type: ignore[arg-type]
    281     # error: Argument 1 to "_na_arithmetic_op" has incompatible type
    282     # "Union[ExtensionArray, ndarray[Any, Any]]"; expected "ndarray[Any
 ↪Any]"
--> 283     res_values = _na_arithmetic_op(left, right, op)  # type:␣
 ↪ignore[arg-type]
    285 return res_values

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/ops/
 ↪array_ops.py:227, in _na_arithmetic_op(left, right, op, is_cmp)
    219 except TypeError:
    220     if not is_cmp and (
    221         left.dtype == object or getattr(right, "dtype", None) == object
    222     ):
   (…)
    225         # Don't do this for comparisons, as that will handle complex␣
 ↪numbers
    226         #  incorrectly, see GH#32047
--> 227         result = _masked_arith_op(left, right, op)
    228     else:
    229         raise

File ~/anaconda3/envs/pandas/lib/python3.12/site-packages/pandas/core/ops/
 ↪array_ops.py:182, in _masked_arith_op(x, y, op)
    179         mask = np.where(y == 1, False, mask)
    181     if mask.any():
--> 182         result[mask] = op(xrav[mask], y)
    184 np.putmask(result, ~mask, np.nan)
    185 result = result.reshape(x.shape)  # 2D compat

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Let's observe average price of cars for each grouping by body-style and drive-wheels

[168]:
```
df.groupby(["body-style","drive-wheels"])["price"].transform('mean')
```

[168]:
```
0      26344.560000
1      26344.560000
2      15706.089474
3      10924.022807
4      13912.066667
           …
200    23883.016667
201    23883.016667
202    23883.016667
203    23883.016667
204    23883.016667
Name: price, Length: 205, dtype: float64
```

[169]:
```
df["average-price"]=df.groupby(["body-style","drive-wheels"])["price"].
 ↪transform('mean')
# selecting columns body-style,drive-wheels,price and average-price
df.loc[:,["body-style","drive-wheels","price","average-price"]]
```

[169]:
```
      body-style drive-wheels    price  average-price
0    convertible          rwd  14844.5   26344.560000
1    convertible          rwd  18150.0   26344.560000
2      hatchback          rwd  18150.0   15706.089474
3          sedan          fwd  15345.0   10924.022807
4          sedan          4wd  19195.0   13912.066667
..           …           …       …            …
200        sedan          rwd  18529.5   23883.016667
201        sedan          rwd  20949.5   23883.016667
202        sedan          rwd  23633.5   23883.016667
203        sedan          rwd  24717.0   23883.016667
204        sedan          rwd  24887.5   23883.016667

[205 rows x 4 columns]
```

3.2 Pivot Tables and Cross-Tabulations

The pandas.pivot_table() function creates a spreadsheet-style pivot table as a dataframe. The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the resulting dataframe.

[170]:
```
new_dataset1 = df.filter(["body-style","drive-wheels",
                          ␣
 ↪"length","width","height","curb-weight","price"],axis=1)
#simplest pivot table with dataframe df and index body-style
table = pd.pivot_table(new_dataset1, index =["body-style"])
table
```

[170]:

body-style	curb-weight	height	length	price	width
convertible	2801.666667	51.433333	170.383333	24079.550000	65.583333
hardtop	2810.625000	52.850000	176.937500	24429.350000	66.612500
hatchback	2348.185714	52.108571	166.592857	11055.314286	65.247143
sedan	2610.593750	54.337500	177.585417	15877.021875	66.260417
wagon	2784.240000	56.728000	181.304000	13609.156000	66.256000

[171]:
```
#pivot table with dataframe df and index body-style and drive-wheels
table = pd.pivot_table(new_dataset1, index =["body-style","drive-wheels"])
table
```

[171]:

body-style	drive-wheels	curb-weight	height	…	price	width
convertible	fwd	2254.000000	55.600000	…	12754.500000	64.200000
	rwd	2911.200000	50.600000	…	26344.560000	65.860000
hardtop	fwd	2008.000000	53.300000	…	9073.900000	63.800000
	rwd	2925.285714	52.785714	…	26622.985714	67.014286
hatchback	4wd	2646.500000	53.850000	…	11445.500000	65.850000
	fwd	2181.551020	52.442857	…	9236.026531	64.671429
	rwd	2746.526316	51.063158	…	15706.089474	66.668421
sedan	4wd	2573.000000	54.300000	…	13912.066667	65.733333
	fwd	2298.228070	53.887719	…	10924.022807	65.326316
	rwd	3108.305556	55.052778	…	23883.016667	67.783333
wagon	4wd	2617.500000	57.000000	…	10005.325000	64.500000
	fwd	2464.333333	56.008333	…	10997.066667	65.533333
	rwd	3284.888889	57.566667	…	18693.644444	68.000000

[13 rows x 5 columns]

[172]:
```
# import numpy for aggregration function
import numpy as np

#new data set with few columns
new_dataset3 = df.filter(["body-style","drive-wheels","price"],axis=1)

# pivot table with dataset new_dataset2
# values are the column in which aggregration function is to be applied
# index is column for grouping of data
# columns for specifying category of data
# aggfunc is the aggregration function to be applied
# fill_value to fill missing values
table = pd.pivot_table(new_dataset3, values='price', index=["body-style"],
                       columns=["drive-wheels"],aggfunc=np.mean,fill_value=0)
table
```

[172]:
```
drive-wheels            4wd           fwd           rwd
body-style
convertible        0.000000  12754.500000  26344.560000
hardtop            0.000000   9073.900000  26622.985714
hatchback      11445.500000   9236.026531  15706.089474
sedan          13912.066667  10924.022807  23883.016667
wagon          10005.325000  10997.066667  18693.644444
```

[173]:
```
table = pd.pivot_table(new_dataset1, values=['price','height','width'],
                       index =["body-style","drive-wheels"],
                       aggfunc={'price': np.mean,'height': [min, max],'width':␣
 ↪[min, max]},
                       fill_value=0)
table
```

[173]:

		height		price	width	
		max	min	mean	max	min
body-style	drive-wheels					
convertible	fwd	55.6	55.6	12754.500000	64.2	64.2
	rwd	53.0	48.8	26344.560000	70.5	64.1
hardtop	fwd	53.3	53.3	9073.900000	63.8	63.8
	rwd	55.4	51.6	26622.985714	72.0	65.0
hatchback	4wd	55.7	52.0	11445.500000	67.9	63.8
	fwd	56.1	49.4	9236.026531	66.6	60.3
	rwd	54.8	49.6	15706.089474	72.3	64.0
sedan	4wd	54.3	54.3	13912.066667	66.4	65.4
	fwd	56.1	50.6	10924.022807	71.4	62.5
	rwd	56.7	47.8	23883.016667	71.7	61.8
wagon	4wd	59.1	54.9	10005.325000	65.4	63.6
	fwd	59.8	53.0	10997.066667	71.4	63.6
	rwd	58.7	54.1	18693.644444	70.3	66.5

Cross-Tabulations

[174]:
```
# apply pd.crosstab() function in data frame df
pd.crosstab(df["make"], df["body-style"])
```

[174]:

body-style	convertible	hardtop	hatchback	sedan	wagon
make					
alfa-romero	2	0	1	0	0
audi	0	0	1	5	1
bmw	0	0	0	8	0
chevrolet	0	0	2	1	0
dodge	0	0	5	3	1
honda	0	0	7	5	1
isuzu	0	0	1	3	0
jaguar	0	0	0	3	0

mazda	0	0	10	7	0
mercedes-benz	1	2	0	4	1
mercury	0	0	1	0	0
mitsubishi	0	0	9	4	0
nissan	0	1	5	9	3
peugot	0	0	0	7	4
plymouth	0	0	4	2	1
porsche	1	2	2	0	0
renault	0	0	1	0	1
saab	0	0	3	3	0
subaru	0	0	3	5	4
toyota	1	3	14	10	4
volkswagen	1	0	1	9	1
volvo	0	0	0	8	3

[175]:
```
# apply margins and margins_name attribute to displays the row wise
# and column wise sum of the cross table
pd.crosstab(df["make"], df["body-style"],margins=True,margins_name="Total Made")
```

[175]:

body-style	convertible	hardtop	hatchback	sedan	wagon	Total Made
make						
alfa-romero	2	0	1	0	0	3
audi	0	0	1	5	1	7
bmw	0	0	0	8	0	8
chevrolet	0	0	2	1	0	3
dodge	0	0	5	3	1	9
honda	0	0	7	5	1	13
isuzu	0	0	1	3	0	4
jaguar	0	0	0	3	0	3
mazda	0	0	10	7	0	17
mercedes-benz	1	2	0	4	1	8
mercury	0	0	1	0	0	1
mitsubishi	0	0	9	4	0	13
nissan	0	1	5	9	3	18
peugot	0	0	0	7	4	11
plymouth	0	0	4	2	1	7
porsche	1	2	2	0	0	5
renault	0	0	1	0	1	2
saab	0	0	3	3	0	6
subaru	0	0	3	5	4	12
toyota	1	3	14	10	4	32
volkswagen	1	0	1	9	1	12
volvo	0	0	0	8	3	11
Total Made	6	8	70	96	25	205

[176]:
```
pd.crosstab([df["make"],df["num-of-doors"]],␣
↪[df["body-style"],df["drive-wheels"]],
```

```
            margins=True,margins_name="Total Made")
```

[176]:

body-style		convertible		hardtop		…	wagon			Total Made
drive-wheels		fwd	rwd	fwd	rwd	…	4wd	fwd	rwd	
make	num-of-doors					…				
alfa-romero	two	0	2	0	0	…	0	0	0	3
audi	four	0	0	0	0	…	0	1	0	5
	two	0	0	0	0	…	0	0	0	2
bmw	four	0	0	0	0	…	0	0	0	5
	two	0	0	0	0	…	0	0	0	3
chevrolet	four	0	0	0	0	…	0	0	0	1
	two	0	0	0	0	…	0	0	0	2
dodge	?	0	0	0	0	…	0	0	0	1
	four	0	0	0	0	…	0	1	0	4
	two	0	0	0	0	…	0	0	0	4
honda	four	0	0	0	0	…	0	1	0	5
	two	0	0	0	0	…	0	0	0	8
isuzu	four	0	0	0	0	…	0	0	0	2
	two	0	0	0	0	…	0	0	0	2
jaguar	four	0	0	0	0	…	0	0	0	2
	two	0	0	0	0	…	0	0	0	1
mazda	?	0	0	0	0	…	0	0	0	1
	four	0	0	0	0	…	0	0	0	7
	two	0	0	0	0	…	0	0	0	9
mercedes-benz	four	0	0	0	0	…	0	0	1	5
	two	0	1	0	2	…	0	0	0	

```
3
mercury       two                0   0       0   0  …     0   0   0
1
mitsubishi    four               0   0       0   0  …     0   0   0
4
              two                0   0       0   0  …     0   0   0
9
nissan        four               0   0       0   0  …     0   3   0
9
              two                0   0       1   0  …     0   0   0
9
peugot        four               0   0       0   0  …     0   0   4
11
plymouth      four               0   0       0   0  …     0   1   0
4
              two                0   0       0   0  …     0   0   0
3
porsche       two                0   1       0   2  …     0   0   0
5
renault       four               0   0       0   0  …     0   1   0
1
              two                0   0       0   0  …     0   0   0
1
saab          four               0   0       0   0  …     0   0   0
3
              two                0   0       0   0  …     0   0   0
3
subaru        four               0   0       0   0  …     2   2   0
9
              two                0   0       0   0  …     0   0   0
3
toyota        four               0   0       0   0  …     2   1   1
18
              two                0   1       0   3  …     0   0   0
14
volkswagen    four               0   0       0   0  …     0   1   0
8
              two                1   0       0   0  …     0   0   0
4
volvo         four               0   0       0   0  …     0   0   3
11
Total Made                       1   5       1   7  …     4  12   9
205

[42 rows x 14 columns]
```

[177]:
```
# rename the columns and row index for better understanding of crosstab
pd.crosstab([df["make"],df["num-of-doors"]],␣
 ↪[df["body-style"],df["drive-wheels"]],
            rownames=['Auto Manufacturer', "Doors"],
            colnames=['Body Style', "Drive Type"],
            margins=True,margins_name="Total Made").head()
```

[177]:
```
Body Style              convertible     hardtop      … wagon         Total
Made
Drive Type                      fwd rwd     fwd rwd  …   4wd fwd rwd
Auto Manufacturer Doors                              …
alfa-romero       two             0   2       0   0  …     0   0   0
3
audi              four            0   0       0   0  …     0   1   0
5
                  two             0   0       0   0  …     0   0   0
2
bmw               four            0   0       0   0  …     0   0   0
5
                  two             0   0       0   0  …     0   0   0
3

[5 rows x 14 columns]
```

[178]:
```
# values are the column in which aggregration function is to be applied
# aggfunc is the aggregration function to be applied
# round() to round the output

pd.crosstab(df["make"], df["body-style"],values=df["curb-weight"],
            aggfunc='mean').round(0)
```

[178]:
```
body-style     convertible  hardtop  hatchback   sedan   wagon
make
alfa-romero         2548.0      NaN     2823.0     NaN     NaN
audi                   NaN      NaN     3053.0  2720.0  2954.0
bmw                    NaN      NaN        NaN  2929.0     NaN
chevrolet              NaN      NaN     1681.0  1909.0     NaN
dodge                  NaN      NaN     2132.0  2056.0  2535.0
honda                  NaN      NaN     1970.0  2289.0  2024.0
isuzu                  NaN      NaN     2734.0  2040.0     NaN
jaguar                 NaN      NaN        NaN  4027.0     NaN
mazda                  NaN      NaN     2254.0  2361.0     NaN
mercedes-benz       3685.0   3605.0        NaN  3731.0  3750.0
mercury                NaN      NaN     2910.0     NaN     NaN
mitsubishi             NaN      NaN     2377.0  2394.0     NaN
nissan                 NaN   2008.0     2740.0  2238.0  2452.0
peugot                 NaN      NaN        NaN  3143.0  3358.0
```

plymouth	NaN	NaN	2208.0	2090.0	2535.0
porsche	2800.0	2756.0	3072.0	NaN	NaN
renault	NaN	NaN	2460.0	NaN	2579.0
saab	NaN	NaN	2724.0	2767.0	NaN
subaru	NaN	NaN	2137.0	2314.0	2454.0
toyota	2975.0	2585.0	2370.0	2338.0	2708.0
volkswagen	2254.0	NaN	2221.0	2342.0	2563.0
volvo	NaN	NaN	NaN	3023.0	3078.0

[179]:
```
# top ten output that represents the percentage of occurrence of the␣
 ↪combination
pd.crosstab(df["make"], df["body-style"],normalize=True).head(10)
```

[179]:

body-style	convertible	hardtop	hatchback	sedan	wagon
make					
alfa-romero	0.009756	0.000000	0.004878	0.000000	0.000000
audi	0.000000	0.000000	0.004878	0.024390	0.004878
bmw	0.000000	0.000000	0.000000	0.039024	0.000000
chevrolet	0.000000	0.000000	0.009756	0.004878	0.000000
dodge	0.000000	0.000000	0.024390	0.014634	0.004878
honda	0.000000	0.000000	0.034146	0.024390	0.004878
isuzu	0.000000	0.000000	0.004878	0.014634	0.000000
jaguar	0.000000	0.000000	0.000000	0.014634	0.000000
mazda	0.000000	0.000000	0.048780	0.034146	0.000000
mercedes-benz	0.004878	0.009756	0.000000	0.019512	0.004878

[ ]:
```

```